A Hardware/Software Architecture for Tool Path Computation. An Application to Turning Lathe Machining

Sergio Cuenca, Antonio Martínez, Antonio Jimeno, Jose Luis Sánchez
Computer Science Technology and Computation Department
University of Alicante
Apdo. Correos 99
03080 Alicante, Spain
email: {sergio, amartinez, jimeno, sanchez}@dtic.ua.es

***Abstract*—Tool path generation is one of the most complex problems in Computer Aided Manufacturing. Although some efficient strategies have been developed, most of them are only useful for standard machining. The algorithm called *Virtual Digitizing* avoids this problem by its own definition but its computing cost is high and makes it difficult for being integrated in standard machining in order to adopt the new ISO standard 14649. Presented in the paper there is a Virtual Digitizing hardware/software architecture that takes advantage of Field Programmable Gate Arrays (FPGAs) to improve the algorithm efficiency and to meet the actual restrictions of the traditional Computer Numeric Control systems at the same time. In order to evaluate the architecture, a prototype was implemented using a commercial reconfigurable platform integrated within a CNC lathe for shoe last machining. The performance of the system for tool path generation was measured for different trajectory and surface precisions using a database of real shoe models. The experiments show a significant speedup for all the cases and maintaining the error of the results below the maximum allowed.**

I. INTRODUCTION

In order to machine a surface by means of a cutting tool on a Computer Numeric Control (CNC) machine tool, a series of 3D or 2D coordinates that define its motion must be supplied. These points are usually referred to as tool centre positions. In this way, the problem can be expressed as *obtaining a trajectory of tool centres that defines the desired object to be machined with a given precision,* in literature the problem is also known as *the tool compensation problem* [1].

With a given object and tool, a solution cannot always be found because of the curvature of the surfaces [2]. In these cases, the problem is redefined in order to obtain a trajectory that defines the closest surface that contains the desired object (that is, without collision). Figure 1 shows the trajectory (tool path) of a circle centre point in order to define a surface. In this case, for the sake of simplicity, the problem is presented in 2D. For 3D surfaces the problem becomes more complex.

Partial solutions to this problem use surface offsets generated by different methods [2,3,4]. However, these offset-surfaces are restricted to one-radius tools (i.e spherical, cylindrical and conical) and are not valid for more complex tools, such as toroidal ones with two radii. Moreover, in most cases, self-intersection problems arise according to the surface curvature. Thus, more sophisticated and higher cost computing techniques are needed to detect and solve these problems.

The Virtual Digitizing (VD) algorithm [5] computes the tool path by means of a "virtually digitized" model of the surface and a geometry specification of the tool and its motion, so can be used even in non-standard machining (retrofitting). This algorithm was developed by one member of our research group and is included in commercial shoe last CAD/CAM software called Forma3D® (from the Spanish Footwear Research Institute, INESCOP). This software is currently a world leader in the CAD/CAM software for shoe lasts. The Virtual Digitalization is simple, robust and avoids the problem of tool-surface collision by its own definition, but its computational is higher than the other approaches.

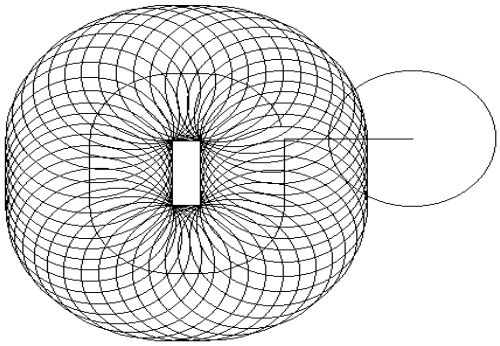

Fig.1. Circle trajectory in order to get a rectangle

On the other hand, the idea of integrating trajectory generation into the numerical control itself is now becoming more common. The new ISO standard 14649 (also called STEP-NC) remedies the shortcomings of ISO 6983 by specifying the machining processes rather than machine tool motion by means of machining tasks.

Unfortunately, in traditional industrial fields, the use of low-performance computers and standard operating systems is widespread. In fact, this is a restriction, since they share both

the management tasks and those of the CAD/CAM. Due to the specific features of the footwear industry and the high cost of new turning lathe machines, the life cycle of these is unusually high (above 20 years). This is specially true in the retrofitting machining, that is, old but efficient hydraulic machines that were modified to be computerly controlled at low cost, substituting the hydraulic system by the electronic one, and introducing low cost PCs for control tasks. For that kind of machines it has been estimated that Pentium I/II based machinery comprises around 85% in the footwear industry. In addition, the use of a low clock rate in the system is a must to improve the inmunity to the electromagnetic interferences generated by their electromechanical parts. This design constrain joined to the cost of the machinery amortization makes the use of modern processors unfeasible at the moment.

In [6, 7] some specific methods for shoe lasts turning lathe machines are proposed. However, their computing cost is high and no specific hardware approach is used for tool path computation. In this work, we propose the use of specific hardware to accelerate the trajectory generation in the control itself and to facilitate the adoption of the new standard by the traditional CNCs.

A study about the theoretical impact of Reconfigurable Computing on the VD algorithm was performed in [8]. A proposal for hardware implementation of the transformation operator used in the VD algorithm can be seen in [9] with some simulation results. In the present work we propose the architecture of the whole hardware/software system and a real implementation has been carried out in order to evaluate the its viability. Using this approach, complex calculations can be made in real time without replacing the numerical control computer and keeping a low clock rate; only a FPGA-based coprocessing board would have to be added.

The paper is divided in the following parts: In section 2 the generic algorithm is explained. In Section 3 a hardware/software architecture is proposed to implement it efficiently. Section 4 summarizes the experiments in order to probe the goodness of the architecture.

The architecture has been succesfully tested in the generation of helicoidal trajectories but, due to the algorithm implemented, it can be used in any other machining environment.

II. VIRTUAL DIGITIZING STRATEGY

With the virtual digitising approach the centre tool points are obtained by virtually touching the object to mechanise . This algorithm typically used to compute pencil curve tracing [10], internally works as mechanical copiers do: the copying arm touches the surface and a group of arms transmitted the movement to the cutting wheels which perform the same movement and finished the copied model.

Due to the fact that all the machining processes are simulated, this algorithm has no restrictions in tool or machine specifications, so the algorithm can be used even in non-standard machining (e.g. in retro-fitting machining).

The digitalization algorithm becomes simple once the surface and tool motion are well defined. Basically, the behavior can be described as follows: For each point of the trajectory the part surface is transformed in order to face the cutting tool. Then the minimum distance from every surface point to the tool is computed in the direction of tool attack axis. This distance determines the tool center point for the current step in the virtual digitalization process. Physically, we select the point that touches the tool surface in first place when the tool is moved along the attack axis.

The basic pseudo-code algorithm can be expressed as follows:

```
For every trajectory position Toolpos do
 Min_dist=∞
 For u in Surface_Rows do
   For v in Surface_Columns do
    p'(u,v) = p(u,v) * TR4x4
    Distance=D(p',Tool)
    If Distance<Min_dist
    then Min_dist=Dist
    Endif
   Endfor
 Endfor
Tool_centre=Centre_point(Min_dist,Toolpos,TR4x4)
Add_trajectory(Tool_centre)
EndFor
```

Fig.2. Basic virtual digitising algorithm

Analyzing algorithm, it is possible to observe up to three nested loops. One of them, the most internal one, is used to access to every surface point in the selected surface, that is, it consists into two loops, one for rows and the other for columns in fact. The most external loop goes through every trajectory position. In order to obtain a good finishing quality, it is necessary produce, at least, as many trajectory points as points the surface has.

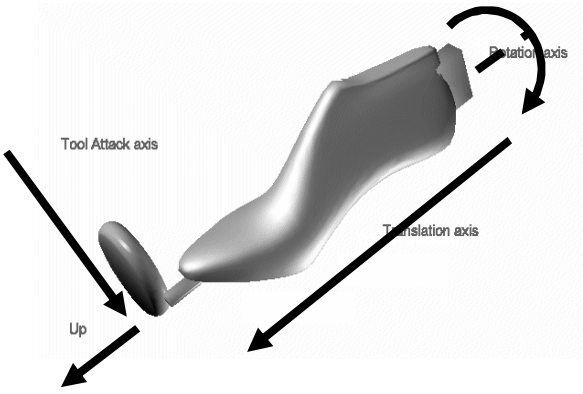

Fig. 3. Axis implied on a shoe last turning lathe machine

Let assume n as the maximum number of surface points, and m as the number of trajectory positions, then the cost of the

algorithm, is: O(m·n).

Values for n and m depend on the model size and the precision desired for the machining. Note that n value consists of a grid of Surface_Rows x Surface_Columns in size for the Algorthim. The more grid points used in each dimension to represent a surface, the finer the spatial resolution of our discretization and the more accurate our trajectory.

As a guide, usual values in shoe last machining can be 10 x 10 x 200 mm, and a grid of approximately 130 x 120 points is used, which implies a distance of 2 mm between points in each surface dimension. Fig 3 shows the axes implied for shoe last machining. All of them perform a spiral movement around the object to be machined. The tool selected in this example is a 3D torus that simulates the cutting tool in movement.

III. HARDWARE/SOFTWARE ARCHITECTURE

On observing the Basic Virtual Digitizing algorithm explained above, we notice that most of the complexity resides inside the third loop. By accelerating the functions called in this part, the total computation time can be significantly reduced.

There are three different operations inside the third loop:

Point transformation : $p'(u,v) = p(u,v) * TR_{4x4}$

A 3D transformation is applied to every surface point according to the selected strategy, so that the tool faces the surface. This operation is made by means of a 4x4 transformation matrix. From a generic standpoint, the process consists of a row * matrix post multiplying. In turning lathe machining this transformation consists of a simple rotation of every point on the rotation axis.

Distance computing: *D(p',Tool)*

This function computes the distance between a surface point and the tool in the tool attack direction. Depending on the complexity of the tool geometry - sphere, torus, cone, and so on - the function becomes more complex. For example, the distance function computes the distance between a 3D point and a 3D torus in the tool attack direction (Y axis) and can be expressed in equation 1.

$$D(x,y,z) = T_y - y - \sqrt{\left(R + \sqrt{r^2 - \left(x - T_x\right)^2}\right)^2 - z^2} \quad (1)$$

Where:

- T_x, T_y are the x,y coordinates of the torus centre
- x,y,z are the 3D point coordinates
- R, r are the major and minor torus radii

Comparison and assignment: If Distance<Min_dist then Min_dist = Distance

Finally, the third nested loop makes a comparison and an assignment if the computed distance is shorter than the current minimum distance.

These three different operations are carried out on every point of the tool trajectory and for every grid point on the original surface. Any optimization made at this level will significantly improve the total computation time.

As expressed above, distance computing implementation varies on tool geometry and point transformation depends on tool path strategy. So if we create hardware circuits (as ASICs) in order to speed up the algorithm for each function, we will need as many circuits as different strategies or tools we are going to use, that is, an expensive and complex architecture. A smart solution would be the use of reconfigurable circuits.

Figure 4 shows our proposed reconfigurable architecture used to perform tool trajectories with virtual digitizing. Different machines, tools and tool path strategies will imply different operation cores for point transformation and distance calculation functions. So GPU (General Purpose Unit) will choose the task involved at a time and reconfigure the RCU (Reconfigurable Co-processor Unit) with the appropriated configuration files stored in the Configuration Memory. Using the partial reconfiguration facilities provided by FPGA devices [11], only a few cores are needed to be maintained in the Conf. Memory while the framework of the algorithm keeps in the RCU.

Surface data and results of processing are stored in a shared memory in order to facilitate the data transactions between both units. An additional channel is used for controlling and configuring the RCU.

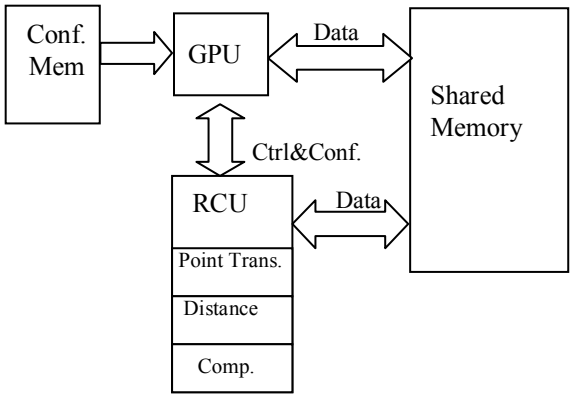

Fig. 4. Virtual Digitizing Architecture

Because the data dependencies between the three functions, the strategy adopted to get the best performance is to unroll the loop by means of a pipeline with three stages, one for every operation. At the same time, every stage will be segmented in several sub-stages to get a well balanced pipeline.

IV. PROTOTYPE IMPLEMENTATION

In order to validate the architecture a prototype has been implemented using the Celoxica RC1000PP PCI board [12]. The board is populated with one Xilinx Virtex1000 FPGA and four 512Kx32b memory banks which can be accessed in

parallel. The local memory of the board is shared with the host, and the transactions of large data blocks can be performed by means of high speed DMA channels. Also there are two additional ports to send commands to the FPGA (Control port) and receive status words in the host (Status port).

Following the segmentation strategy, all operations in the inner loops could be included in one sufficiently large FPGA and executed in a pipeline fashion. However, the distance calculation, due to its floating point nature may be a difficult and resource consuming task for implementing on this version of the Virtex devices. On the other side, the point transformation operation can be easily migrated to fixed point arithmetic. For this reason we propose, as first approach, to use the FPGA to perform the point transformation and to leave the distance calculation in the charge of the host microprocessor. The algorithm partition is shown in figure 5.

```
//GPU task
While ToolPos < MaxXPos do
 Receive Surface_TR_Vectors[NSteps]
 For n in NSteps do
  For every col in Surface_TR_Vectors[n]do
    Distance=D(p',Tool)
    If Distance<Min_dist
    then Min_dist=Dist
    Endif
  Endfor
  Tool_centre=Centre_point(Min_dist,Toolpos,TR4x4)
  Add_trajectory(Tool_centre)
 Endfor
EndWhile
```

```
//RCU task
While ToolPos < MaxXPos do
 For n in NSteps do
  For u in Surface_Rows do
   For v in Surface_Columns do
    If p(u,v)∈Tool_Influence_Area
     p'(u,v) = p(u,v) * TR4x4
     Add_Surface_TR_Vectors[n](p')
    Endif
   Endfor
  Endfor
  Increment(ToolPos)
 EndFor
 Send Surface_TR_Vectors[NSteps]
EndWhile
```

Fig. 5. Partition of the basic algorithm

Where:

- *X axis* is the translation axis of the tool (so the last position in the tool trajectory).
- *MaxXPos* is the maximum value of x coordinate in the surface points.
- *Tool_Influence_Area* is the area of influence for the tool in every trajectory position, that is the subset of the 3D surface points that the tool could reach in its actual position.
- *Surface_TR_Vectors[n]* represents the vector n that comprises the set of transformed points in the *Tool_Influence_Area* for a specific *ToolPos*. The number of transformed points that composes a vector depends of the position of the tool.
- *NSteps* is the number of RCU iterations executed prior to send the data to the GPU. In every hardware step a transformed surface vector is generated.
- *Send* and *Receive* primitives include the mechanism and synchronization methods to transfer blocks of data between the GPU and the RCU.

Celoxica HandelC [13] high level hardware description language was chosen for implementing the hardware side of the system. This C-based HDL allowed the use of the VD algorithm, originally coded in C, as starting point for porting it to hardware. Although compilation from HandelC descriptions will not equal the performance of hand-coded VHDL, the turnaround time to get those first results working may be an order of magnitude better. After this initial phase, it was possible to iteratively modify and recompile HandelC code to obtain higher performance. Such design iterations took only a matter of minutes, whereas the same iterations may require hours of even days if we were used VHDL or Verilog.

For the co-simulation of the whole system, the Celoxica Data Stream Manager [14] (DSM) was used. This hw/sw API simplifies the transfer of data between the GPU and the RCU, this way all the components of the design were tested working together to ensure correct functionality. Thanks to DSM the verification of the system was carried out with the original testbench of the software version of the algorithm.

Two levels of segmentation were used to implement the architecture.

A. GPU-RCU segmentation

The first level of segmentation allows overlapping the tasks of both parts of the system. While RCU computes the Surface_TR_Matrix for the iteration (i), the GPU works on the Surface_TR_Matrix for the iteration (i-1). Every iteration involves the execution of NSteps surface transformations in the RCU and the generation of NSteps trajectory positions in the GPU.

The synchronization between the two threads (GPU task and RCU task) is performed by means of two RC1000 library functions that allow the access to the Control and Status ports. Both functions are blocking and only return when the read or write operation has completed.

As figure 6 shows, the GPU starts the algorithm writing the data surface to the memory banks 0 and 1 in the RC1000 board. Then it sends the command to the FPGA and keeps waiting the RCU response. Once the FPGA has received the command in the Control port, the pipeline starts its work reading the data surface and writing the transformed points to the banks 2 and 3. This work is performed NSteps times, generating one transformed surface vector every time. When the pipeline ends, the FPGA send the data valid code through the Status port. Next, the host read the vector descriptors (indicating the number of components of every vector) to prepare the DMA transference of the Surface_TR_Vectors.

Once the DMA finish the host acknowledges the transaction through the Control port. At this point the FGPA starts the second iteration while the host begins with the completion of the first one. A new synchronization will be place when both of them finish their work.

```
//GPU task
Pack(Surface);
Do_DMA_Write(Surface,ram0,ram1)
PP1000WriteControl(START)
While ToolPos < MaxXPos do
 PP1000ReadStatus() //block until FPGA write
 Do_DMA_Read(Vectors_Descriptor,ram2)
 Do_DMA_Read(Surface_TR_Vectors,ram2,ram3)
 PP1000WriteControl(ACK) // FPGA starts new ite
 Unpack (Surface_TR_Vectors)
 // Process the Surface_TR_Vectors to get
 // the trajectory points
 ....
EndWhile
```

```
//RCU task
PP100ReadControl() //block until host write
While ToolPos < MaxXPos do
 For n in NSteps do
  Process(Surface,ram0,ram1,ram2,ram3)
 Endfor
 PP1000WriteStatus(DATA_VAL)
 PP1000ReadControl // block until host write
Endfor
```

Fig. 6. Synchronization of threads

B. *RCU segmentation*

The second level of segmentation performs the unrolling of the internal loops to accelerate the execution within the RCU. In order to get the best performance, the points of the surface were packed into 64bit words, coding every coordinate in a 21bit fixed point number. This way was possible to access one point in a single clock cycle reading banks 0 and 1 in parallel. In a similar way, the transformed points can be written to the banks 2 and 3, in only one cycle.

To pack and unpack the data sent and received to/from the RCU, several software functions was designed and added to the GPU code. The resultant RCU pipeline is composed by 5 stages where the products of TR are performed by multipliers synthesized by the HandelC compiler. Its throughput is 1point/cycle.

The error introduced by the 21bit fixed point approximation was analyzed, and as a result, an absolute error, in the worst case, equal to 2^{-11} was obtained. This value is lower than the maximum error allowed in the tool trajectory, which is 0,1mm.

V. EXPERIMENTS

For test purposes the VD algorithm was firstly implemented in ANSI-C and compiled with MS Visual Studio 2005 Express using the best optimization level supported by the compiler (similar to level 3 in gcc compilers).

Next the system was tested with a large data base of shoe lasts to get the execution times of the software version for further comparisons. Two configurations was employed for the test: system A equipped with a "state of the art" microprocessor (Pentium IV @3,2 GHz) will be used as reference to explore the limits of the RCU; system B (Pentium I @166MHz) represents the state of the art of CNC lathe in shoe last machining and will be used for speedup measures.

In all the experiments the absolute error obtained for the tool trajectory points was always under 0,1mm as the previous analysis had predicted.

The FPGA resources consumed by the core of the RCU were 11% of the available Slices, and the maximum frequency predicted by the implementation tool (Xilinx ISE v7.1) was 53,073MHz.

A. *Tuning of the architecture parameters*

The aim of the first set of experiments was to configure the architecture to get the best performance but keeping low the clock rate of the hardware. One of the key parameters is NSteps, since it determines the degree of overlapping between hardware and software stages. If NSteps is too high the duration of the RCU tasks can be greater than the execution time of the GPU, becoming a bottleneck for the system. For its maximum value, equal to the number of trajectory positions of the tool, the hardware and software will work sequentially. On the other hand, if NSteps is too low, the number of DMA transactions will be very high and the transference overhead could be significant for the total computation time.

The other key is the clock rate of the hardware system; obviously the performance is directly related with it but also the robustness of the system due to the electromagnetic interference. Any value below 100MHz is allowed for a real environment.

For the experiments we selected a typical shoe last (131x120 points) and trajectory (0,22 points/mm). Fig. 7 and 8 show the execution time in seconds of the hardware/software architecture for different NSteps values in two different systems. As shown in this figures, the behavior is similar in both graphics. Clearly for low frequencies the hardware side of the system is the bottleneck and limits the performance. The execution time decreases significantly with every increment of the frequency. There is a limit where the time remains stable for a determined NSteps value. This happens around 25MHz for system A and 10MHz for system B. At this point the hardware and software tasks are well balanced. Above this frequency the software became the slowest stage of the system and any improvement of the hardware will not have a noticeable impact on the performance.

The NSteps parameter presents an asymptotic behavior also. For every frequency there is a minimum NSteps value, from it the time does not change significantly. This value is around 18 for system A and near 45 for system B. This could be explained for the difference between the performance in the PCI transactions of both systems (A: 120MB/s; B: 32MB/s). Because of its low performance system B needs a greater reduction in the number of DMA transactions to balance the

impact on the software execution time.

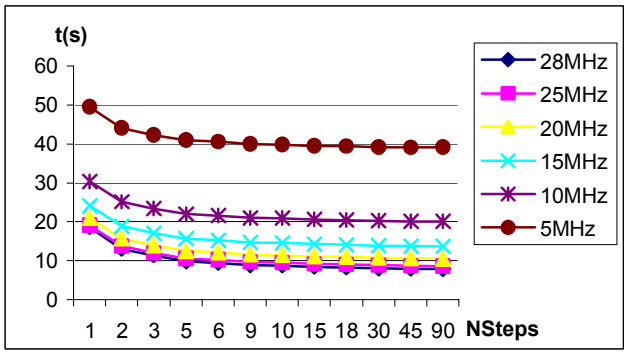

Fig. 7. Execution times for system A: PentiumIV (@3,2MHz, 1GB RAM)

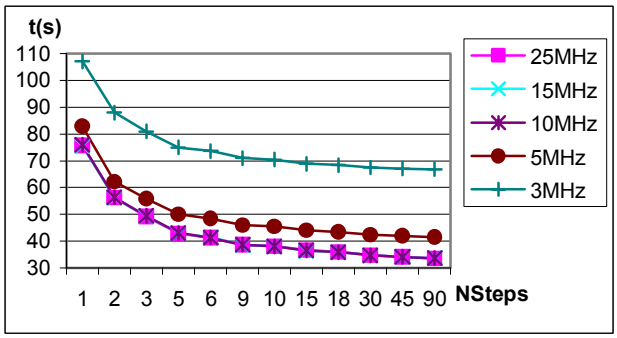

Fig. 8. Execution times for system B: PentiumI (@ 166MHz, 16MB RAM)

The speedup of the best configuration for the system B is 7.05, four times slower than the system A and only 3 times slower than the fully software version running on Pentium IV system. From measures obtained on system A is deduced that there is still a significant room for further improvements to get the limits of the architecture.

B. Performance measures

As seen in section 2, the computing cost associated to a tool path generation algorithm depends on two factors: the precision of the geometric model, in our case a shoe last modeled by a discretization of a NURBS surface, and the precision of the trajectory itself. In order to evaluate the performance of the proposed architecture, we are taken into account both factors. We establish the surface precision "sp" as the average number of surface points per squared millimeter. On the one hand, we define the precision of the trajectory "tp" as the average number of points per millimeter of trajectory length.

Figure 9 and 10 shows the achieved speedup vs the precision factors commented above. Speedup is calculated using, as a base time, the computing time for a fully software version of the same algorithm running on system B. We have selected a value of 90 for the NSteps parameter, and a 28MHz working frequency. As shown in these experiments, the speedup increases when the complexity of the surface and/or the trajectory are increased. This is a positive fact, since the more complex the tool path computing is, the faster our system runs.

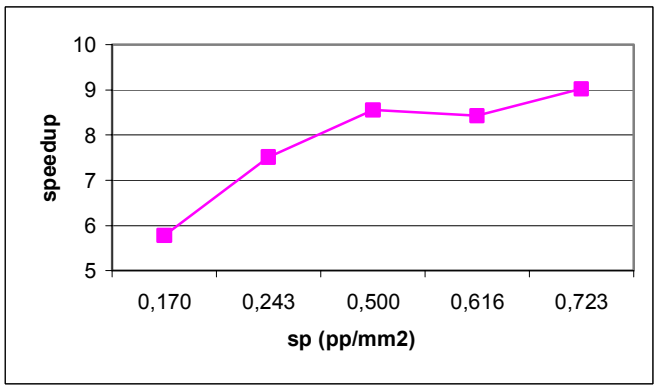

Fig. 9. Speedup versus surface precision

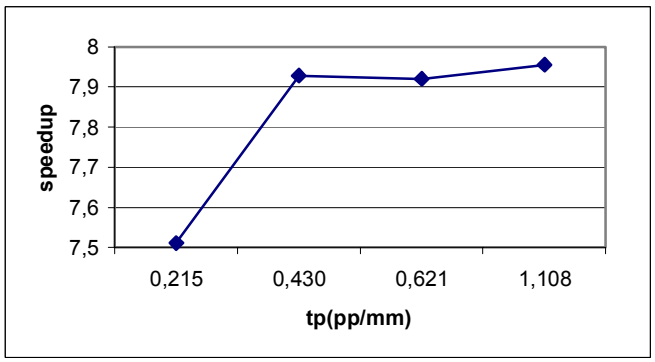

Fig. 10. Speedup versus trajectory precision

VI. CONCLUSIONS

Because of the adoption of the new ISO standard 14649, nowadays is a tendency for numerical controls to incorporate advanced characteristics in trajectory generation.

The virtual digitising algorithm is simple to implement, offers good results and avoids the problem of tool collision by its own definition. However, the algorithm is not suitable for general-purpose machining, since it is too slow compared with other types of tool path generation algorithms. This is especially true for traditional manufacturing environments, such as in the manufacture of shoe lasts where is generalized the use of low performance computers.

A hybrid hardware/software approach has been proposed to solve this problem arising from the industrial CAD/CAM field. A prototype of the architecture has been implemented taken as baseline system a CNC for shoe last machining. The results show a significant increase of the computing speed, maintaining a relatively low frequency at the same time. This fact confirm the feasibility of the proposal, the computational cost could be replaced by the insertion of specialized boards instead of changing the whole control system.

Further refinements can be studied to improve the response time of the system, for instance the adoption of a superscalar architecture for the RCU. In addition, future studies aim to complete the hardware architecture so that it can support the distance calculation task in the virtual digitizing algorithm.

REFERENCES

[1] Farin, G. "Curves and Surfaces for Computer Aided Geometric Design. A Practical Guide". Academic Press Inc., 1993.

[2] Wang, Y. "Intersection of offsets of parametric surfaces". Computer Aided Geometric Design, vol. 13, pp.453-465, 1996.

[3] Choi, B. K. "Surface Modeling for CAD/CAM". Elsevier Science Publishers, pp. 263-272, 1991.

[4] Held, M. "Voronoi Diagrams and Offset Curves of Curvilinear Polygons". Computer-Aided Design, vol. 30, no. 4, pp. 287-300, 1998.

[5] Jimeno A, García J., Salas F. "Shoe lasts machining using Virtual Digitising". International Journal of Advanced Manufacturing Technology. Date of publication June 2001 in Vol 17 No.10.

[6] Chen, J., Gong, Y., Jin, T., Tong, S, "Development of an integrated CAD/CAM system for shoe last", 2005 IEEE International Conference on Mechatronics and Automation, ICMA 2005, pp. 1107-1111.

[7] Denkena, B., Scherger, S., "A concept for shoe last manufacturing in mass customisation". 2005 CIRP Annals - Manufacturing Technology 54 (1), pp. 341-344.

[8] Jimeno, A., Cuenca, S. "Reconfigurable Computing for Tool-Path Computation" International Journal of Advanced Manufacturing Technology, vol. 21, no 12, pp. 945-951, 2003.

[9] Jimeno, A. Sánchez, J.L., Mora, H. Mora J. García-Chamizo, J.M. FPGA-based tool path computation An application for shoe last machining on CNC lathes. Computers in industry Elsevier, 2006, vol. 57, nº2, pp. 103-111.

[10] Jung W Park et al "Pencil Curve Tracing via Virtual Digitizing" Proc. of IFIP CAPE Conference, (1991), pp.97-104.

[11] Two Flows for Partial Reconfiguration: Module Based or Difference Based. Xilinx Application Note XAPP290, version 1.1, Xilinx, Inc. (2003)

[12] Celoxica Ltd. RC1000 Hardware reference manual v2..3. http://www.celoxica.com/cup/registered_users/manuals/default.asp

[13] Celoxica Ltd. Handel C reference manual . http://www.celoxica.com/techlib/files/CEL-W0410251JJ4-60.pdf

[14] Celoxica Ltd. Data Stream Manager Datasheet. http://www.celoxica.com/techlib/files/CEL-W0506201CG6-46.pdf